Rhythmic Tunes - Frontend Development with React.js

1.Introduction

- **Project Title:** Rhythmic Tunes - Your Melodic Companion
- **Team Members names and their roles:**

Sandhiya H (Team Leader)-hsandhiya853@gmail.com
Padma shree B (Team Member)- padmashreebaskaran@gmail.com
Shruthi C (Team Member) - krcshruthi@gmail.com
Sandhiya R (Team Member) – sandhiyasandhiya87555@gmail.com
Thilagavathi M (Team Member) – tthilaga534@gmail.com

2. Project Overview

Purpose:

Music has always been an essential part of human life, offering comfort, entertainment, and emotional connection. With the evolution of digital platforms, users now expect seamless, high-quality music streaming experiences that allow them to access their favorite songs anytime, anywhere. Rhythmic Tunes is a modern, interactive music streaming application built using React.js, designed to provide a user-friendly and immersive audio experience. The primary goal of this project is to create a feature-rich, intuitive, and responsive platform where users can explore, play, organize, and personalize their music collections with ease. Unlike traditional music players, Rhythmic Tunes focuses on enhancing user engagement through personalized playlists, advanced search capabilities, and smooth navigation. The application ensures that music lovers can discover new songs, save their favorites, create playlists, and enjoy offline listening, making it a comprehensive solution for all music streaming needs. By leveraging modern web technologies, Rhythmic Tunes offers a fast, visually appealing, and accessible experience, ensuring that users can enjoy music without interruptions, whether on a desktop, tablet, or mobile device.

Features:

- Song Listings with details (title, artist, genre, etc.)
- Playlist Creation & Management
- Playback Controls (Play, Pause, Skip, Adjust Volume)
- Offline Listening
- Advanced Search Functionality
- Wishlist/Favorites Management

3. Architecture

Component Structure of Rhythmic Tunes

Rhythmic Tunes follows a modular, component-based architecture in React.js, ensuring scalability, reusability, and maintainability. Below is a breakdown of the main components, their responsibilities, and how they interact.

1. App Component (App.js)

Role: The root component that manages the overall application structure and routing.

Key Responsibilities:

- Sets up React Router for navigation.
- Includes a global layout such as the Navbar and Sidebar.
- Loads different pages based on the current route.

2. Pages (Located in /src/pages/)

a) Songs Page (Songs.js)

Role: Displays a list of available songs with search and playback options.

Key Features:

- Fetches song data from http://localhost:3000/items.
- Allows users to play, pause, and add songs to a playlist or favorites.
- Implements a search bar for filtering songs.

Implementation

Uses useEffect to fetch songs, wishlist, and playlist from JSON Server.

Uses useState to manage currently playing song and search queries

b) Playlist Page (Playlist.js)

Role: Manages user-created playlists.

Key Features:

- Fetches the playlist from http://localhost:3000/playlist.
- Allows adding/removing songs from playlists.
- Provides playback controls for songs in the playlist.

Implementation

Uses useState to track the playlist.

Uses axios.post() to add songs to the playlist and axios.delete() to remove them.

c) Favorites Page (Favorites.js)

Role: Displays songs that users have marked as favorites.

Key Features:

- Fetches favorite songs from http://localhost:3000/favorities.
- Allows removing songs from favorites.

Implementation

Uses useState for wishlist management.

Uses axios.post() to add songs to favorites and axios.delete() to remove them.

3. Core Components (Located in /src/components/)

a) SongCard.js

Role: Displays an individual song with title, artist, and audio controls.

Properties:

- **song:** Object containing song details (title, artist, url).
- **onPlay**: Function to handle audio playback.
- **onAddToPlaylist:** Function to add song to playlist.
- **onAddToFavorites:** Function to add song to favorites.

Implementation

Uses Bootstrap card structure.

Uses <audio> tag for music playback.

Uses FontAwesome icons (FaHeart, FaRegHeart) for wishlist management.

b) AudioPlayer.js

Role: Manages song playback and audio controls.

Features:

- Handles playing/pausing songs.
- Ensures only one song plays at a time.

Implementation

Uses useEffect() to track currently playing song.

Uses an event listener to pause previous songs when a new song starts playing.

c) SearchBar.js

Role: Provides a search input to filter songs.

Properties:

- onSearch(query): Function to filter the song list.

Implementation

Uses Bootstrap’s InputGroup for styling.

Filters songs based on title, artist, or genre.

d) Sidebar.js

Role: Provides navigation links to Songs, Playlist, and Favorites.

Implementation :

Uses React Router's NavLink for navigation.

State Management

State management in Rhythmic Tunes is handled through local state (useState) and global state (Context API), ensuring efficient data handling and smooth user interactions.

Local State (useState) is used for managing temporary UI states such as:

- Search queries entered by the user.
- Currently playing song and playback state.
- Real-time updates when songs are added/removed from the playlist or favorites.

Global State (Context API) is used for:

- Storing playlists and favorites across the application.
- Ensuring updates in one component reflect globally.
- Avoiding excessive prop drilling by centralizing data.

API Data Fetching (useEffect & Axios): The application retrieves data from JSON Server (http://localhost:3000/), ensuring real-time updates for songs, favorites, and playlists.

Routing

Rhythmic Tunes uses React Router (react-router-dom) to handle client-side navigation, ensuring an SPA (Single Page Application) experience without reloading the page. The routes are defined in App.js, linking to different sections of the application:

- "/" → Displays the Songs page.
- "/playlist" → Navigates to User Playlists.
- "/favorites" → Opens Favorites List.

The BrowserRouter component in App.js ensures that users can switch between pages smoothly, making navigation fast and efficient.

Setup Instructions

Setting up the Rhythmic Tunes project involves installing the required software, cloning the repository, and configuring the environment to ensure smooth execution. Before proceeding, ensure that your system has all the necessary dependencies, including Node.js, npm, Git, a code editor, and JSON Server. Node.js and npm are required to run the React.js application, while Git helps in version control and managing the project repository. A code editor like VS Code is recommended for an efficient development experience. Additionally, JSON Server is used to simulate a mock backend for handling API requests and managing song data.

Prerequisites

Before installing the project, ensure that the following software and tools are installed on your system:

1. **Node.js & npm** – Required for running the React.js application.

Download & Install: Node.js Official Website

Check installation:

```
node -v  # Checks Node.js version
npm  -v   # Checks npm version
```

2. Git – For cloning the repository and managing version control.

Download & Install: Git Official Website

Verify installation:

```
git –version
```

3. Code Editor – Any text editor such as VS Code, WebStorm, or Sublime Text.

Recommended: Visual Studio Code

4. JSON Server – To run a mock backend for API testing.

Install globally using npm:

```
npm install -g json-server
```

Installation Steps

Follow these steps to set up the project on your local machine:

1. Clone the Repository

Open a terminal and run the following command:

```
git clone [your-repository-url]
```

Navigate into the project folder:

```
cd rhythmic-tunes
```

2. Install Dependencies

Run the following command to install all required packages:

- npm install

This will install dependencies such as React, React Router, Axios, and Bootstrap required for the project.

3. Configure Environment Variables (if needed)

If the project requires environment variables, create a .env file in the root directory and add necessary API keys or configurations.

Example:

- REACT_APP_API_URL=http://localhost:3000

4. Run the Application

Start the React development server:

- npm run dev # If using Vite

The application will be accessible at:

http://localhost:5173

5. Start the JSON Server (Mock API Backend)

Open a new terminal and navigate to the project folder.

Run the following command to start the JSON server:

- json-server --watch ./db/db.json

This will enable the mock backend and provide API endpoints like:

http://localhost:3000/items

6. Verify the Application

Open a web browser and go to http://localhost:5173 to check if the application is running correctly.

Try searching for a song, adding it to a playlist, and playing it to confirm everything works as expected.

Folder Structure

The Rhythmic Tunes project follows a well-organized folder structure to ensure scalability, maintainability, and efficient development. The React application is structured into different directories, each serving a specific purpose. The main folders include components, pages, assets, and utilities, making it easy to manage and reuse code across the project. This structured approach enhances code readability, modularity, and reusability, allowing developers to easily navigate and modify different parts of the application.

Client (React Application Structure)

The frontend of Rhythmic Tunes is structured as follows:

```
rhythmic-tunes/
│── src/
│   ├── components/      # Reusable UI components
│   │   ├── AudioPlayer.js
│   │   ├── SongCard.js
│   │   ├── SearchBar.js
│   │   ├── Navbar.js
│   │   ├── Sidebar.js
│   ├── pages/          # Main pages/views of the application
│   │   ├── Songs.js
│   │   ├── Playlist.js
│   │   ├── Favorites.js
│   ├── assets/         # Images, icons, and other static files
│   ├── utils/          # Helper functions, custom hooks, API calls
│   ├── context/        # Global state management (if using Context API)
│   ├── App.js          # Main application file
│   ├── index.js        # Entry point of the application
│── public/             # Static assets like index.html
│── package.json        # Dependencies and scripts
│── README.md           # Project documentation
```

Key Folder Descriptions

1. components/

Contains reusable UI components such as the Audio Player, Search Bar, Navbar, Sidebar, and Song Card.

Components are modular, making it easier to maintain and update individual elements without affecting the entire application.

2. pages/

Includes different views of the application, such as:

Songs.js → Displays a list of available songs.

Playlist.js → Manages user-created playlists.

Favorites.js → Lists favorited songs for easy access.

3. assets/

Stores static files such as images, icons, and audio thumbnails.

Helps in keeping the project clean and structured by separating UI elements from logic.

4. utils/

Contains helper functions and utility files to handle repeated logic.

Examples of utilities used in the project:

api.js → Centralized API calls using Axios.

formatTime.js → Converts song durations into a readable format.

customHooks.js → Stores reusable custom React hooks like useFetchSongs().

5. context/ (If using Context API)

Stores global state management logic to manage playlists, favorites, and song playback state across the app.

6. public/

Contains static files, including index.html, which serves as the root file for rendering the React application.

Running the Application

Once the Rhythmic Tunes project is set up, you can run the application locally to test its functionality and user interface. The application consists of a frontend React.js app and a mock backend using JSON Server. Running both ensures that the app fetches and displays songs, playlists, and favorites correctly. Follow the steps below to start the application on your local machine.

Steps to Run the Application Locally

1**. Navigate to the Project Directory**

Open a terminal and move into the project folder:

- cd rhythmic-tunes

2. **Start the Frontend Server**

Run the following command to start the React development server:

```
npm run dev   # If using Vite
```

This will start the application, and you can access it in a browser at:

http://localhost:5173

3. Start the JSON Server (Mock Backend)

Open a new terminal and navigate to the project directory.

Run the following command to start the backend server:

```
json-server --watch ./db/db.json
```

This will create API endpoints at:

http://localhost:3000/items
http://localhost:3000/favorities
http://localhost:3000/playlist

4. Verify the Application

Open a web browser and visit http://localhost:5173.

Test features like searching songs, adding to playlists, and marking favorites to ensure everything works correctly.

Component Documentation

Songs Component

- **Purpose:** Displays the list of available songs.
- **Props:**
  - songList (Array): Contains song details (title, artist, genre).
  - onPlay (Function): Handles song playback.

Player Component

- **Purpose:** Controls audio playback.
- **Props:**
  - currentSong (Object): The song currently playing.
  - onPause (Function): Pauses playback.
  - onNext (Function): Skips to the next song.

State Management

Global State

- Uses React Context API to store user data, playlists, and favourites.
- Example:

```js
const [favourites, setFavourites] = useState([ ]);
```

Local State

- Example:

```js
const [searchTerm, setSearchTerm] = useState(' ');
```

User Interface

- **Modern UI** with a clean layout.
- **Dark & Light Mode** support.
- **Smooth Animations** for enhanced experience.

Styling

- **CSS Frameworks:** Bootstrap & Tailwind CSS.
- **Custom Styles:** Styled-components for reusable design.
- **Theming:** Dark mode toggle implemented.

Testing

- **Jest & React Testing Library** for unit tests.
- **Cypress** for end-to-end testing.
- **Mock API Calls** tested using axios-mock-adapter.

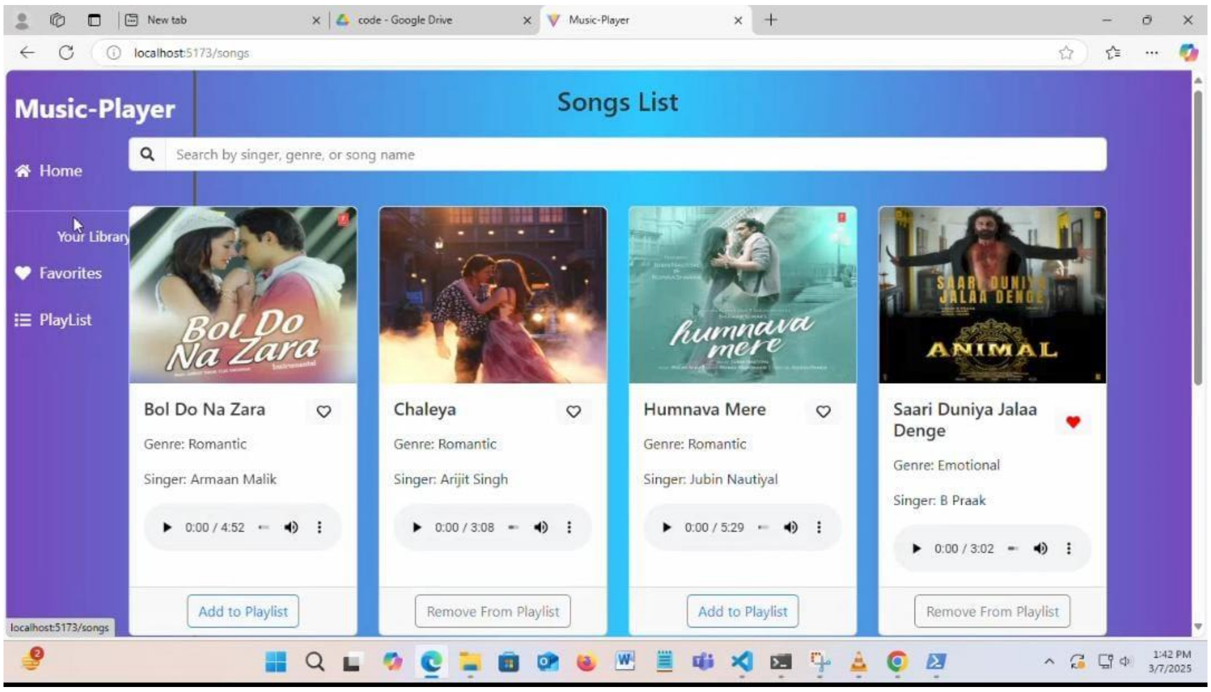
Screenshots or Demo

- **ProjectDemo:**

https://drive.google.com/drive/folders/1bKP8DhSi8gLtJa9O0sUCMJ8iiTsDQLLA

- **Screenshots:**

Home Page:



Songs List

Search by singer, genre, or song name

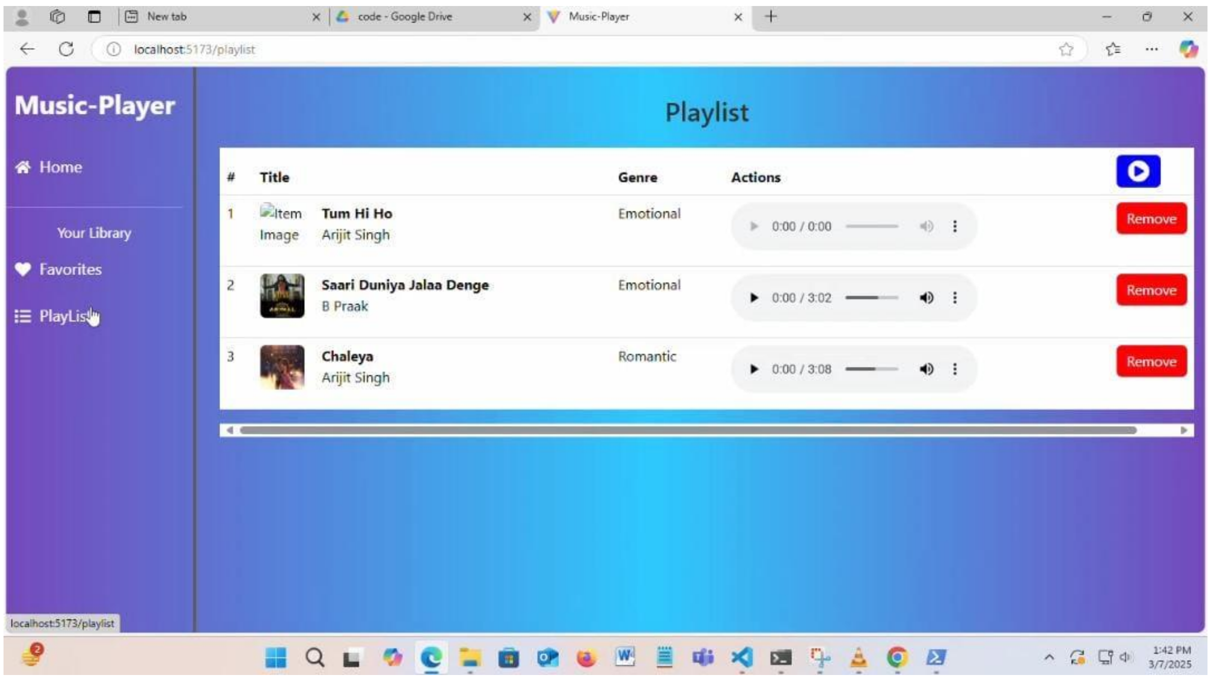
Bol Do Na Zara
Genre: Romantic
Singer: Armaan Malik
0:00 / 4:52
Add to Playlist

Chaleya
Genre: Romantic
Singer: Arijit Singh
0:00 / 3:08
Remove From Playlist

Humnava Mere
Genre: Romantic
Singer: Jubin Nautiyal
0:00 / 5:29
Add to Playlist

Saari Duniya Jalaa Denge
Genre: Emotional
Singer: B Praak
0:00 / 3:02
Remove From Playlist

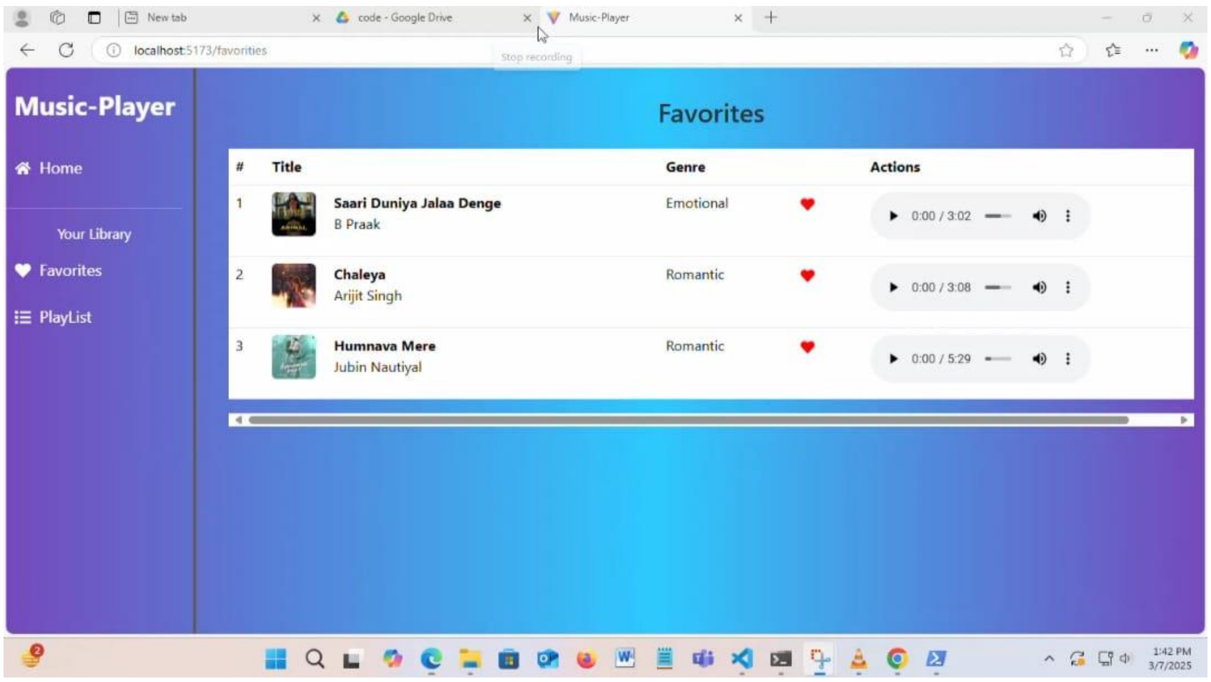
Playlist Management



Playlist

#	Title		Genre	Actions	
1	Item Image	Tum Hi Ho Arijit Singh	Emotional	0:00 / 0:00	Remove
2		Saari Duniya Jalaa Denge B Praak	Emotional	0:00 / 3:02	Remove
3		Chaleya Arijit Singh	Romantic	0:00 / 3:08	Remove

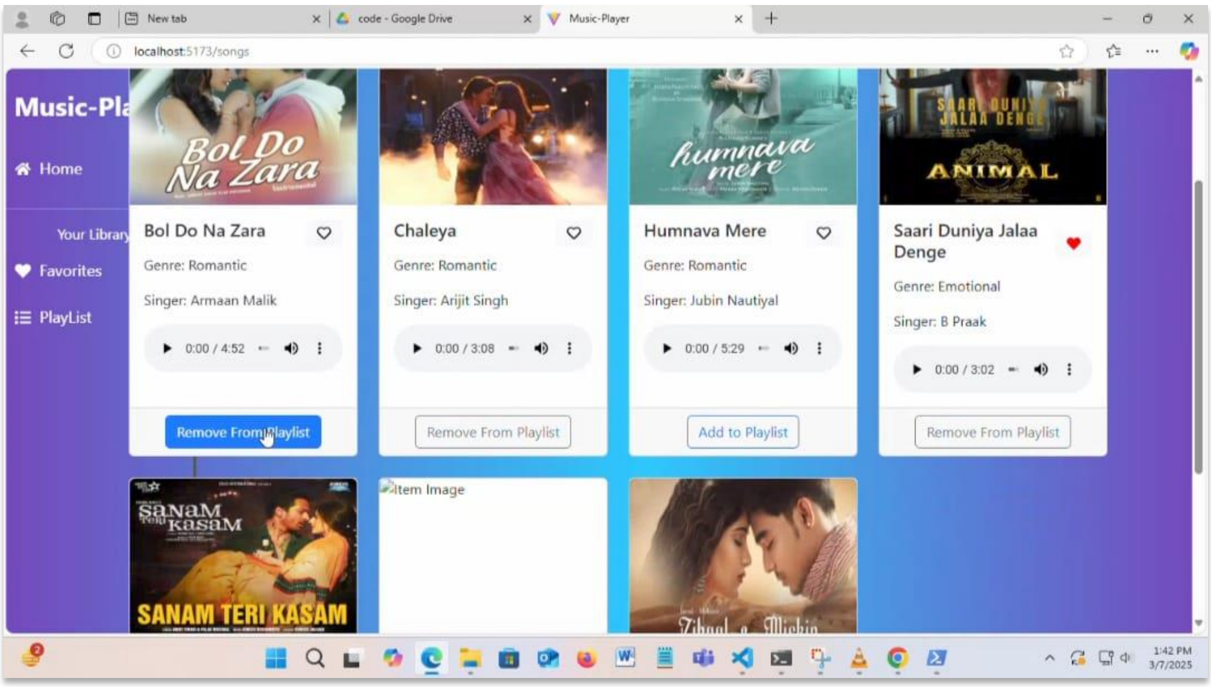
Favorite Songs



Favorites

#	Title	Genre		Actions
1	Saari Duniya Jalaa Denge B Praak	Emotional	♥	0:00 / 3:02
2	Chaleya Arijit Singh	Romantic	♥	0:00 / 3:08
3	Humnava Mere Jubin Nautiyal	Romantic	♥	0:00 / 5:29

Music Player UI



Bol Do Na Zara
Genre: Romantic
Singer: Armaan Malik
0:00 / 4:52
Remove From Playlist

Chaleya
Genre: Romantic
Singer: Arijit Singh
0:00 / 3:08
Remove From Playlist

Humnava Mere
Genre: Romantic
Singer: Jubin Nautiyal
0:00 / 5:29
Add to Playlist

Saari Duniya Jalaa Denge
Genre: Emotional
Singer: B Praak
0:00 / 3:02
Remove From Playlist

Known Issues

- Audio playback may not sync properly across devices.
- Offline feature requires additional caching improvements.

Future Enhancements

- **AI-Powered Song Recommendations** based on user preferences.
- **Podcast Feature** for artist interviews and music insights.
- **Live Lyrics Display** synchronized with the song.
- **Social Sharing** for playlists and favorite songs.